DEPSTAR

Devang Patel Institute of Advance Technology and Research

(A Constitute Institute of CHARUSAT)

Certificate

This is to certify that

Mr. / Mrs. Probin Bhagchandani

of Computer Engineering *Class,*

ID. No. 22DCE006 *has satisfactorily completed*

his/ her term work in Competitive Programming *for*

the ending in Nov 2024/2025

Date : 17/10

A.D.K

Sign. of Faculty

Dgang

Head of Department

CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPSTAR

22DCE006 – Probin Bhagchandani

5CE1

Batch - A

Competitive Programming File

PRACTICAL - 1

PROBLEM DEFINITION:

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

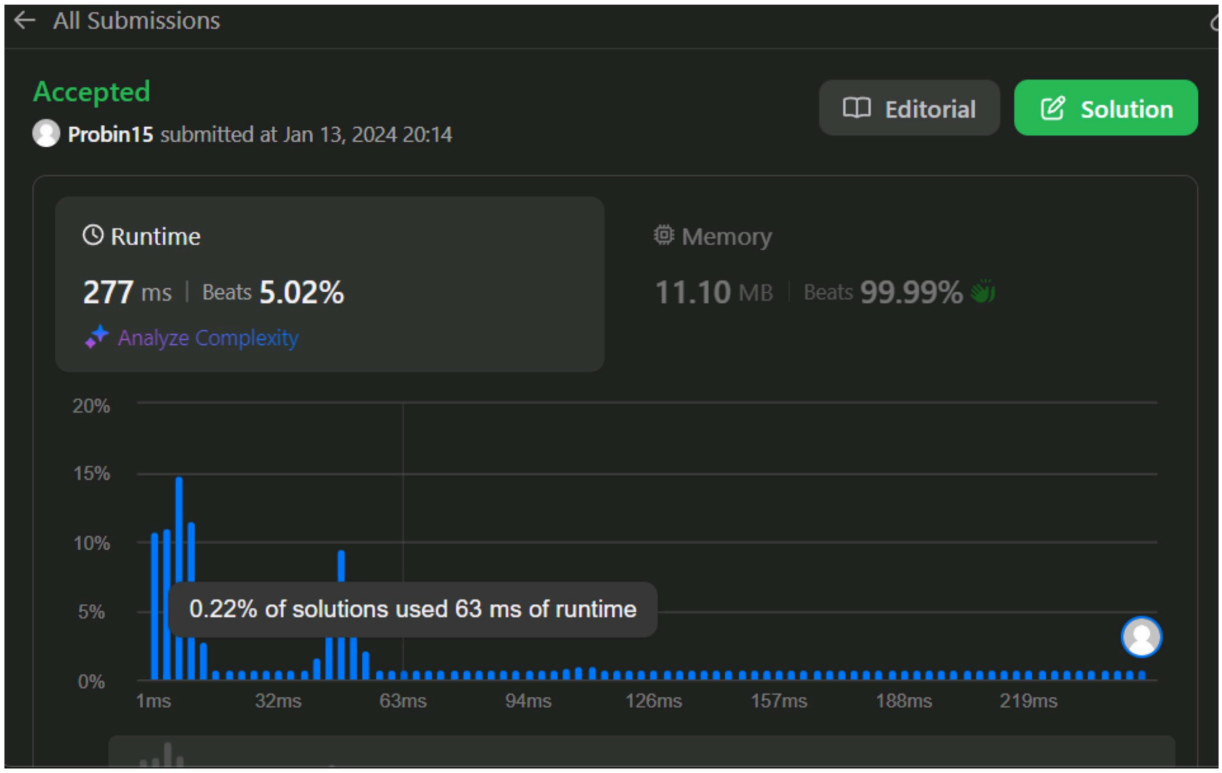
You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

PROGRAM CODE :

```
class Solution(object):

def twoSum(self, nums, target):

ans = []

a=len(nums)

for i in range(0,a,1):

c=i

for j in range(i+1,a,1):

sume=nums[c]+nums[j]

if sume==target:

ans.append(c)

ans.append(j)

return ans
```

OUTPUT :


All Submissions
Accepted
Probin15 submitted at Jan 13, 2024 20:14
Editorial
Solution
Runtime
277 ms | Beats 5.02%
Analyze Complexity
Memory
11.10 MB | Beats 99.99%
20%
15%
10%
5%
0%
0.22% of solutions used 63 ms of runtime
1ms
32ms
63ms
94ms
126ms
157ms
188ms
219ms

CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPSTAR

PRACTICAL - 2

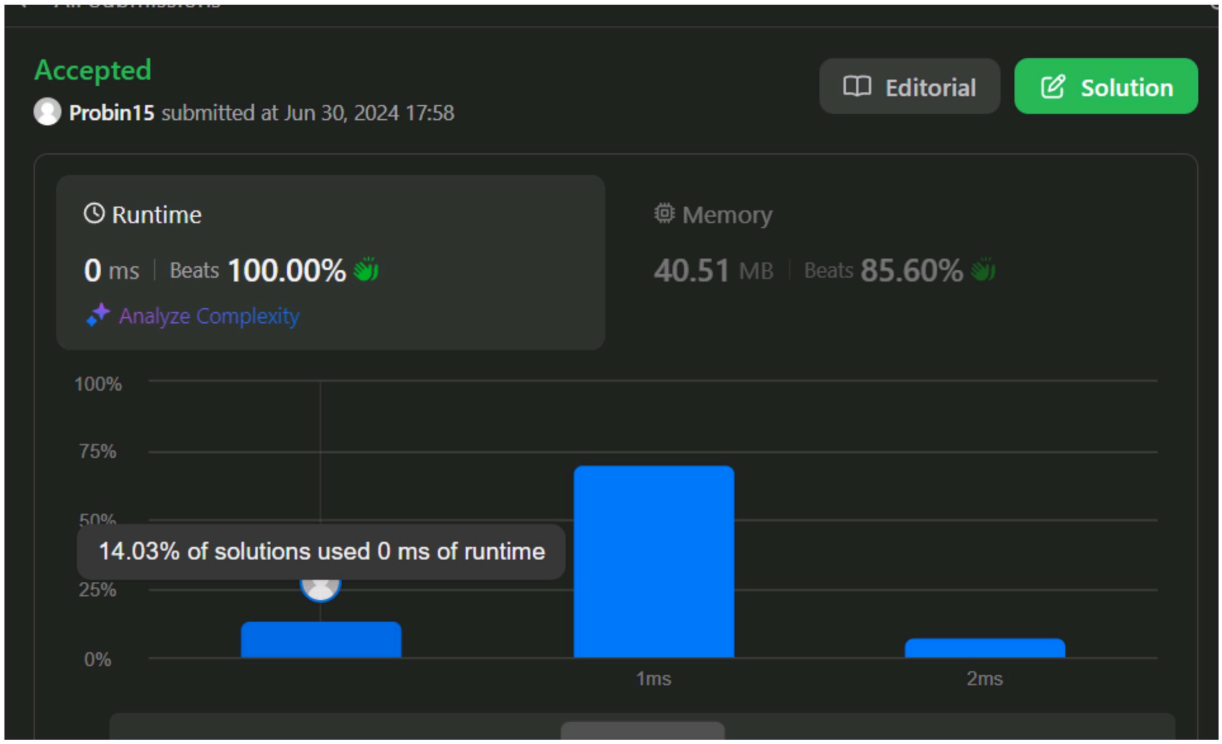
PROBLEM DEFINITION:

Given a 32-bit signed integer x, return x with its digits reversed. If reversing x causes the value to go outside the signed 32-bit integer range [-2^31, 2^31 - 1], then return 0.

Program Code:

```
class Solution {
    public int reverse(int x) {
        long rev = 0;
        while (x != 0) {
            rev = rev * 10 + x%10;
            if (rev > Integer.MAX_VALUE || rev < Integer.MIN_VALUE) {
                return 0;// check range if r is outside the range then return 0
            }
            x = x/10;
        }
        return (int) rev;
    }
}
```

OUTPUT :



PRACTICAL - 3

PROBLEM DEFINITION:

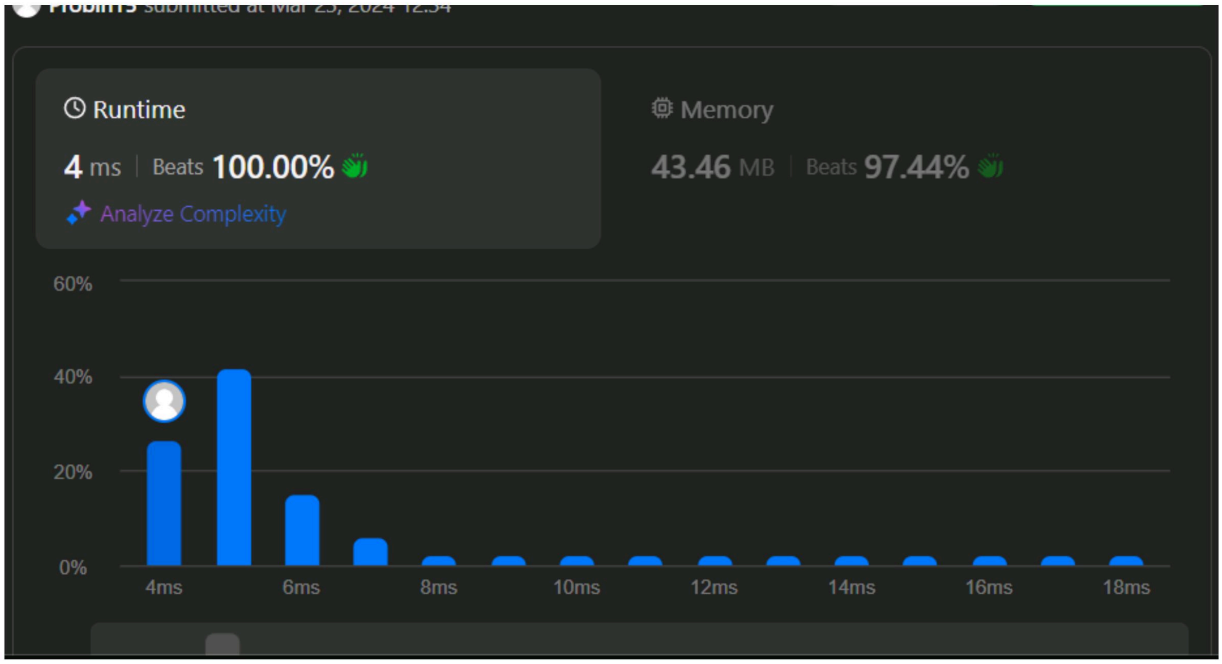
Problem Definition:

Given an integer x, return true if x is a palindrome, and false otherwise.

Program Code:

```
import java.util.*;
public class Solution
{
    public static boolean isPalindrome(int x)
    {
        int num=x;
        int reverse=0;
        int rem;

        while(num>0)
        {
            rem=num%10;
            reverse=(reverse*10 )+ rem;
            num=num/10;
        }
        if(reverse==x)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    public static void main(String args[])
    {
        int x=121;
        isPalindrome(x);
    }
}
```

OUTPUT :


Runtime
4 ms | Beats 100.00%
Analyze Complexity
Memory
43.46 MB | Beats 97.44%
60%
40%
20%
0%
4ms
6ms
8ms
10ms
12ms
14ms
16ms
18ms

PRACTICAL - 4

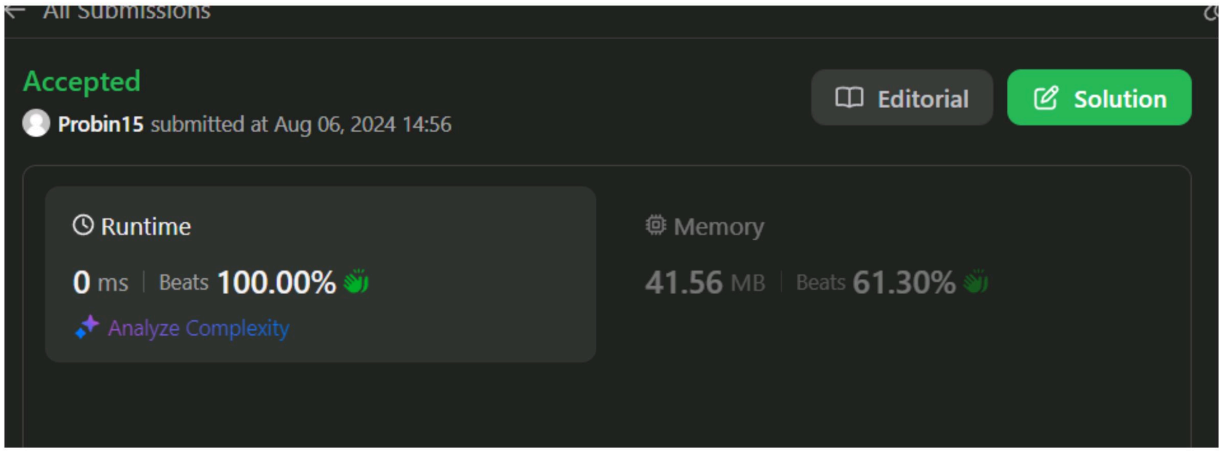
PROBLEM DEFINITION:

Given a sorted array that is rotated at an unknown pivot, search for a target value. If the target is found, return its index; otherwise, return -1. The array has no duplicate elements.

Program Code:

```
class Solution{

    public ListNode removeNthFromEnd(ListNode head, int n) {
        if (head.next == null)
            return null;

        int size = 0;
        ListNode temp = head;
        while (temp != null) {
            size++;
            temp = temp.next;
        }

        if (n == size) {
            head = head.next;
            return head;
        }

        ListNode prev = head;
        for (int i = 1; i < size-n; i++) {
            prev = prev.next;
        }

        ListNode delnode = prev.next;

        if (delnode.next == null){
            prev.next = null;
        }
        else{
            prev.next = delnode.next;
        }

        return head;
    }
}
```

OUTPUT :


All Submissions
Accepted
Probin15 submitted at Aug 06, 2024 14:56
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
41.56 MB | Beats 61.30%

PRACTICAL - 5

PROBLEM DEFINITION:

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Program Code:

```
class Solution {
    public boolean isValid(String s) {
        Stack<Character> stk = new Stack<>();
        int slen = s.length();

        for (int i = 0; i < slen; i++) {
            char ch = s.charAt(i);
            if (ch == '{' || ch == '[' || ch == '(') {
                stk.push(ch);
            } else {
                if (stk.isEmpty()) {
                    return false;
                }
                char top = stk.pop();
                if ((top == '{' && ch != '}') || (top == '(' && ch != ')') ||
                    (top == '[' && ch != ']')) {
                    return false;
                }
            }
        }
        return stk.isEmpty();
    }
}
//opening=[,{,(-push to stack
//closing=}])-check for non empty stack and check if matching closing bracket is
present for the top element of the stack.
```

OUTPUT :

PRACTICAL - 6

PROBLEM DEFINITION:

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

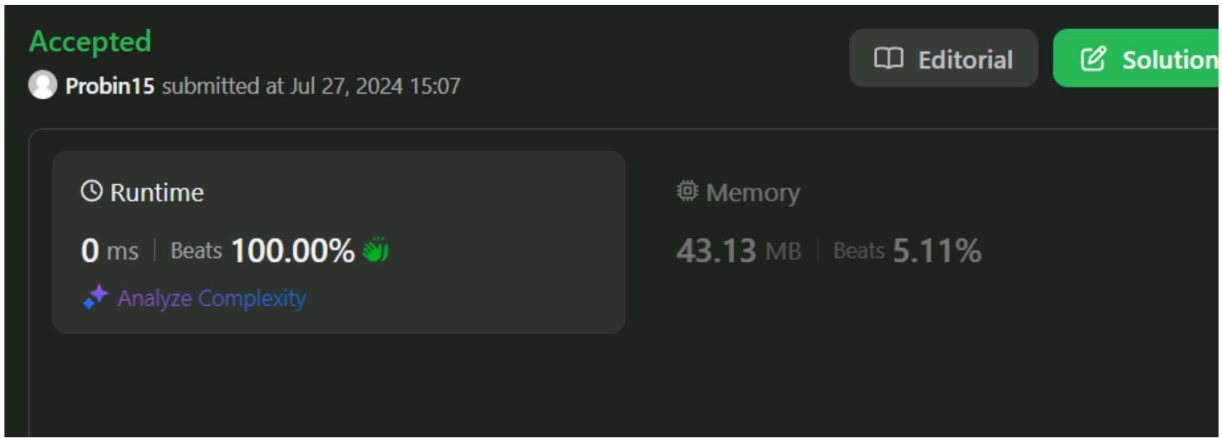
Return *the head of the merged linked list*.

Program Code:

```
class Solution {
```

```
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        if (list1 == null)
            return list2;
        if (list2 == null)
            return list1;

        if (list1.val < list2.val) {
            list1.next = mergeTwoLists(list1.next, list2);
            return list1;
        } else {
            list2.next = mergeTwoLists(list1, list2.next);
            return list2;
        }
    }
}
```

OUTPUT :



PRACTICAL - 7

PROBLEM DEFINITION:

Given an integer array nums and an integer val, remove all occurrences of val in nums in-place. The order of the elements may be changed. Then return *the number of elements in* nums *which are not equal to* val.

Consider the number of elements in nums which are not equal to val be k, to get accepted, you need to do the following things:

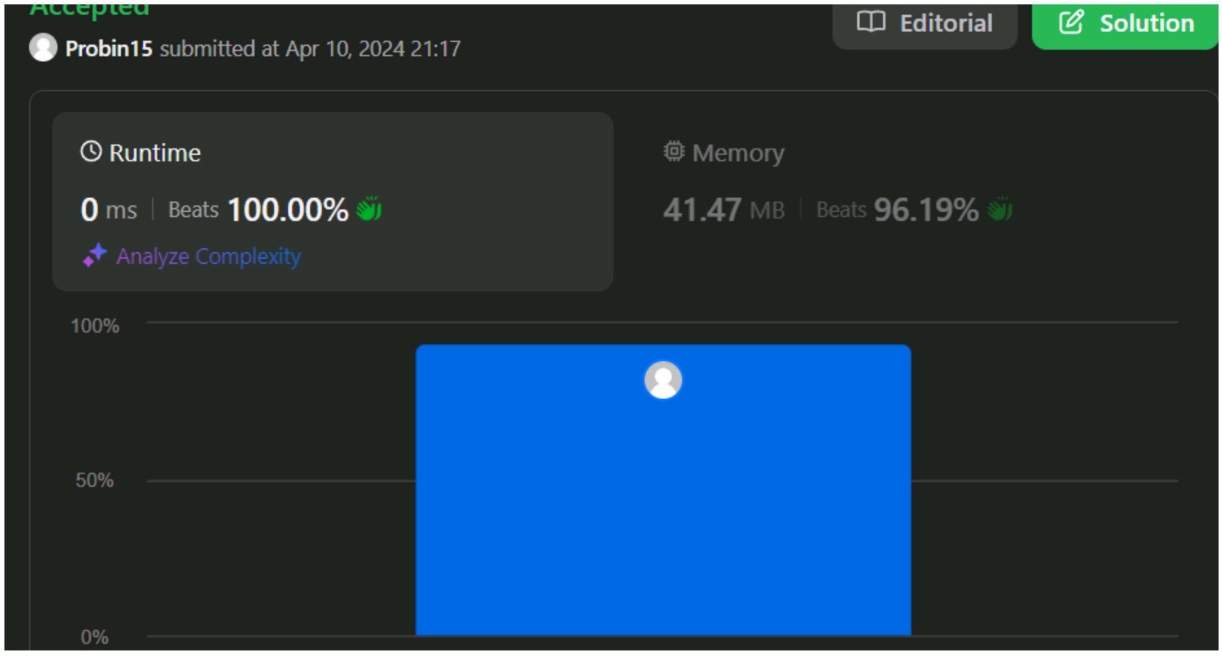
- Change the array nums such that the first k elements of nums contain the elements which are not equal to val. The remaining elements of nums are not important as well as the size of nums.

- Return k.

Program Code:

```
class Solution {
    public int removeElement(int[] nums, int val) {
        int i = 0;
        int count = 0;
        for (int j = 0; j < nums.length; j++) {
            if (nums[j] != val) {
                nums[i] = nums[j];
                i++;
                count++;
            }
        }

        return count;
    }
}
```

OUTPUT :



PRACTICAL - 8

PROBLEM DEFINITION:

There is an integer array nums sorted in ascending order (with **distinct** values).

Prior to being passed to your function, nums is **possibly rotated** at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (**0-indexed**). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

Given the array nums **after** the possible rotation and an integer target, return *the index of* target *if it is in* nums, *or* -1 *if it is not in* nums.

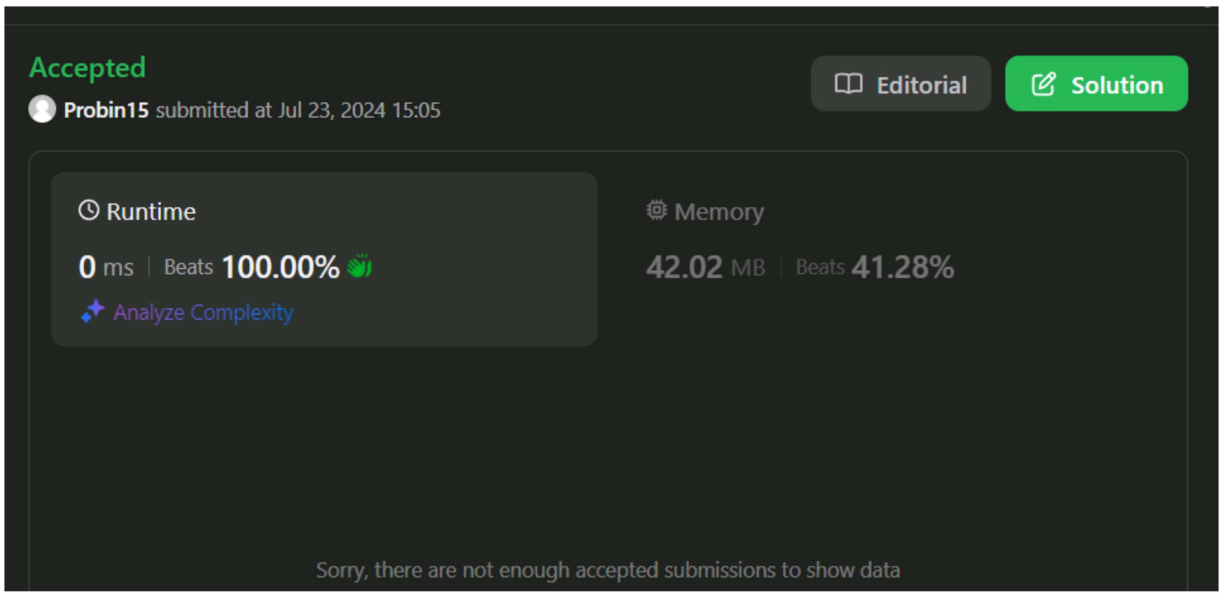
You must write an algorithm with O(log n) runtime complexity.

Program Code:

```
class Solution {
    public int search(int[] nums, int target) {
        int start=0;
        int end=nums.length-1;
        while(start<=end)
        {
            int mid=(start)+(end-start)/2;

            if(nums[mid]==target)
                return mid;
            else if(nums[start]<=nums[mid]){

                if(target>=nums[start]&&target<=nums[mid]){

                    end = mid-1;
                }
                else{
                    start = mid+1;
                }
            }
            else{

                if(target>=nums[mid] && target<=nums[end]){
                    start = mid+1;
                }
                else{
                    end = mid-1;
```

```
        }
    }

    }
    return -1;
  }
}
```

OUTPUT :



PRACTICAL - 9

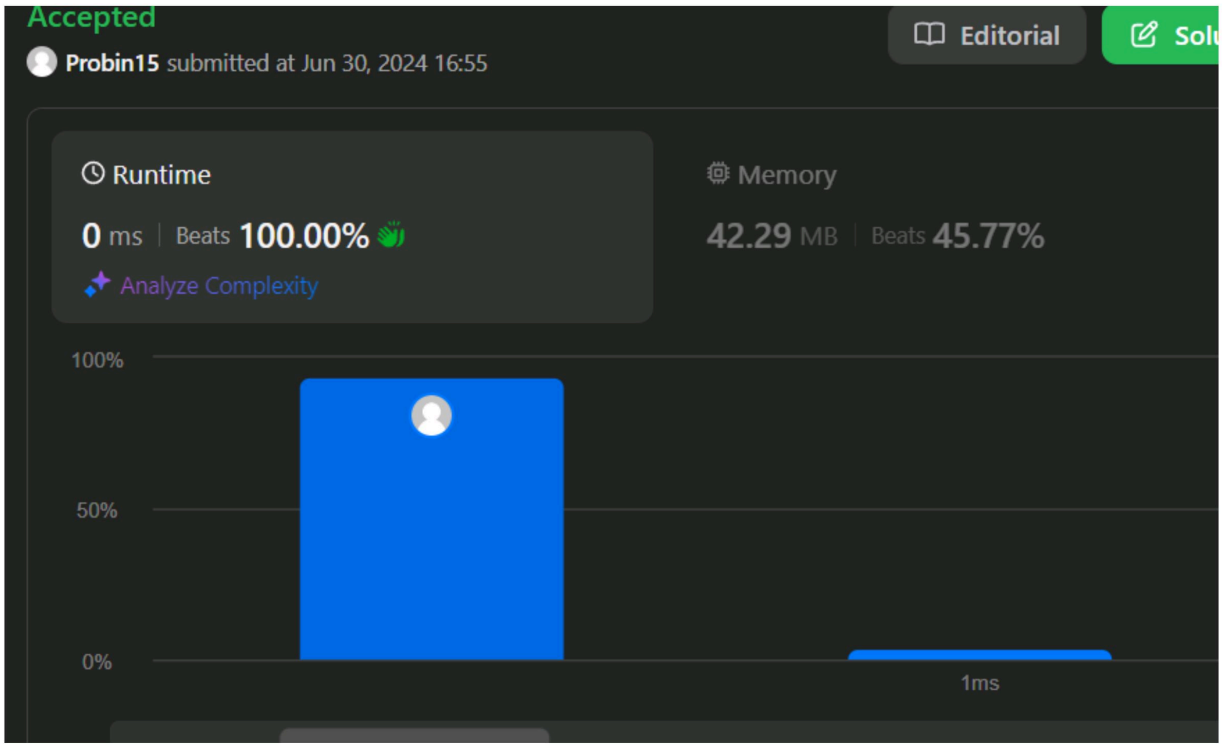
PROBLEM DEFINITION:

Implement pow(x, n), which calculates x raised to the power n (i.e., x^n).

Program Code:

```
class Solution {
    public double myPow(double x, int n) {
        if (x == 0) {
            return 0;
        }
        if (n == 0) {
            return 1;
        }
        double y = myPow(x, n / 2);
        double ans = y * y;
        if (n % 2 != 0) {
            if (n < 0) {
                ans = (1 / x) * ans;
            } else {
                ans = x * ans;
            }
        }
        return ans;
    }
}
```

OUTPUT :



PRACTICAL - 10

PROBLEM DEFINITION:

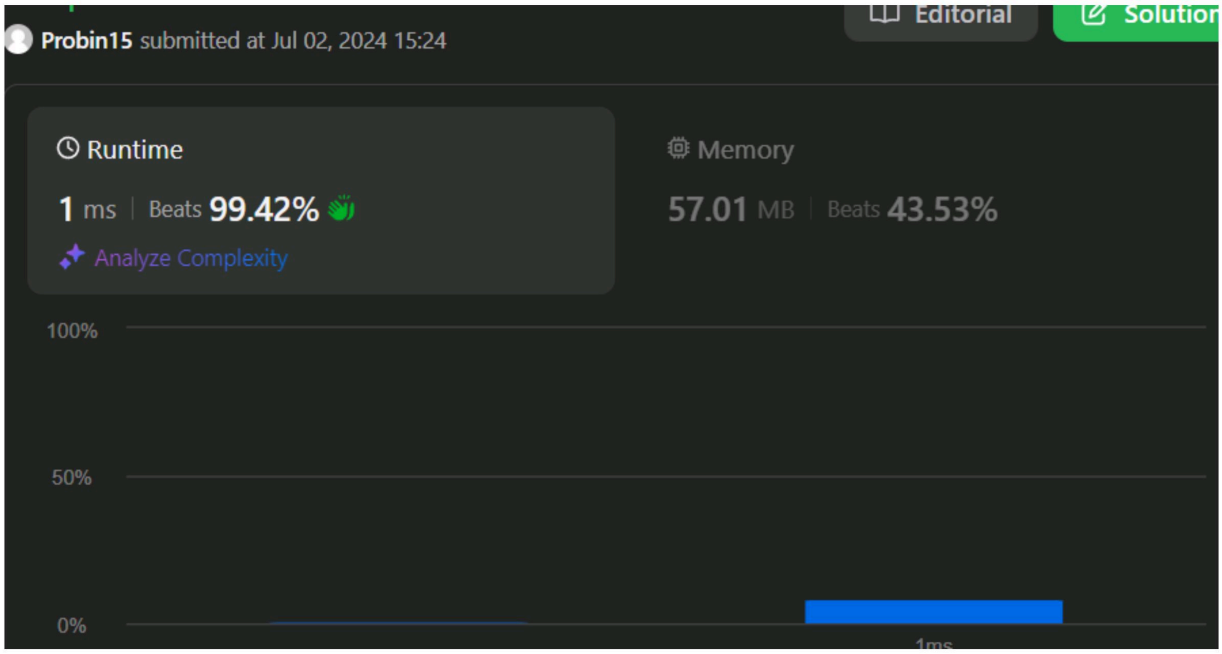
Given an integer array nums, find the subarray with the largest sum, and return *its sum*.

Program Code:

```
class Solution
{
    public int maxSubArray(int[] nums)
    {
        int maxSum = Integer.MIN_VALUE;
        int sum = 0;
        for(int i=0;i<nums.length;i++)
        {
            sum = sum+nums[i];
            maxSum = Math.max(sum,maxSum);
            if(sum<0) sum = 0;
        }
        return maxSum;
```

```
    }
}
```

OUTPUT :



PRACTICAL - 11

PROBLEM DEFINITION: You are given an integer array nums. You are initially positioned at the array's **first index**, and each element in the array represents your maximum jump length at that position.

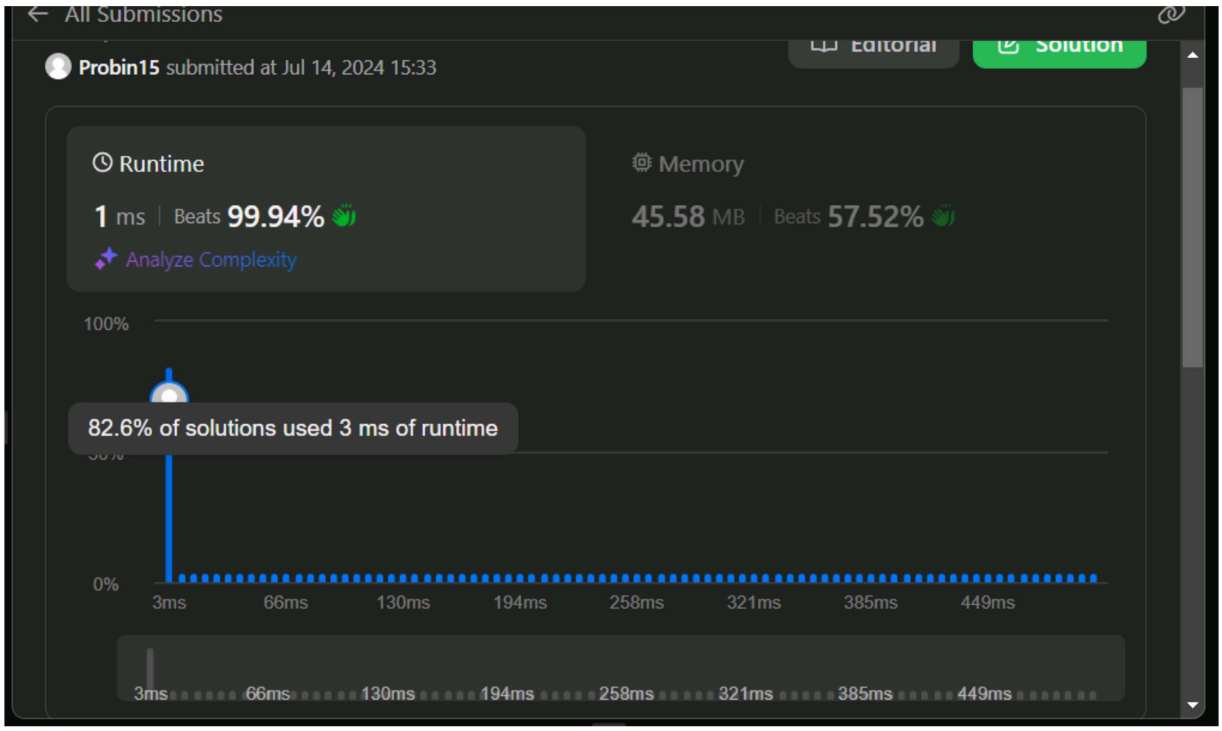
Return true *if you can reach the last index, or* false *otherwise*.

Program Code:

```
class Solution {
    public boolean canJump(int[] nums) {
        int target = nums.length - 1;
        for (int i = nums.length - 2; i >= 0; i--) {
            if (target <= i + nums[i]) {
                target = i;
            }
```

```
        }
        return target == 0;
    }
}
```

OUTPUT :



PRACTICAL - 12

PROBLEM DEFINITION:

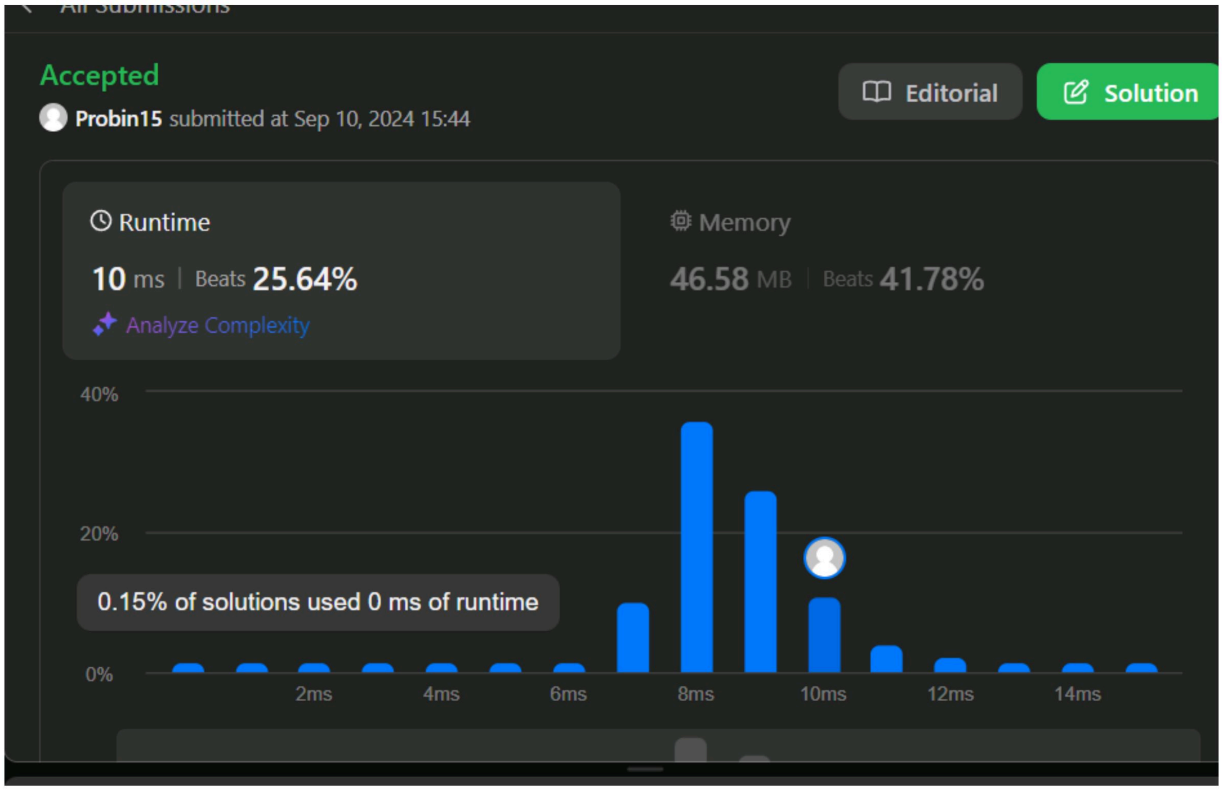
Given an array of intervals where intervals[i] = [start, end], merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input.*

Program Code:

```
class Solution {
    public int[][] merge(int[][] intervals) {
        if (intervals.length <= 1) {
            return intervals;
        }

        Arrays.sort(intervals, Comparator.comparingInt(i -> i[0]));
        ArrayList<int[]> result = new ArrayList<>();
```

```
        int[] newInterval = intervals[0];
        result.add(newInterval);

        for (int[] interval : intervals) {
            if (interval[0] <= newInterval[1]) {
                newInterval[1] = Math.max(newInterval[1], interval[1]);
            } else {
                newInterval = interval;
                result.add(newInterval);
            }
        }
        return result.toArray(new int[result.size()][]);
    }
}
```

OUTPUT :



PRACTICAL - 13

PROBLEM DEFINITION:

Given a string s consisting of words and spaces, return *the length of the **last** word in the string.*

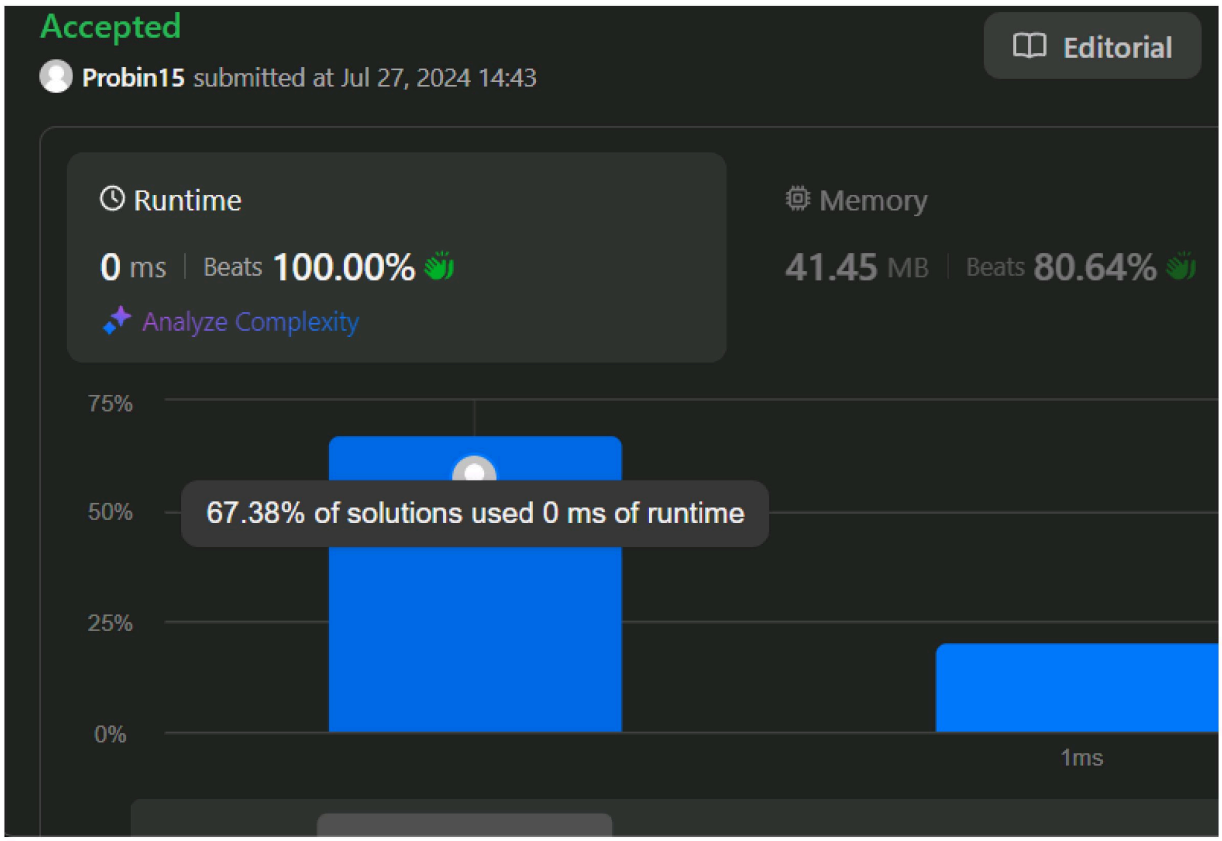
A **word** is a maximal

substring
consisting of non-space characters only.

Program Code:

```
class Solution {
    public int lengthOfLastWord(String s) {
        int len = 0;
        s = s.trim();
        for (int i = s.length() - 1; i >=0; i --){
            if (s.charAt(i) == ' '){
                break;
            }
            else{
                len ++;
            }
        }
        return len;
    }
}
```

OUTPUT :



PRACTICAL - 14

PROBLEM DEou are given a **large integer** represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

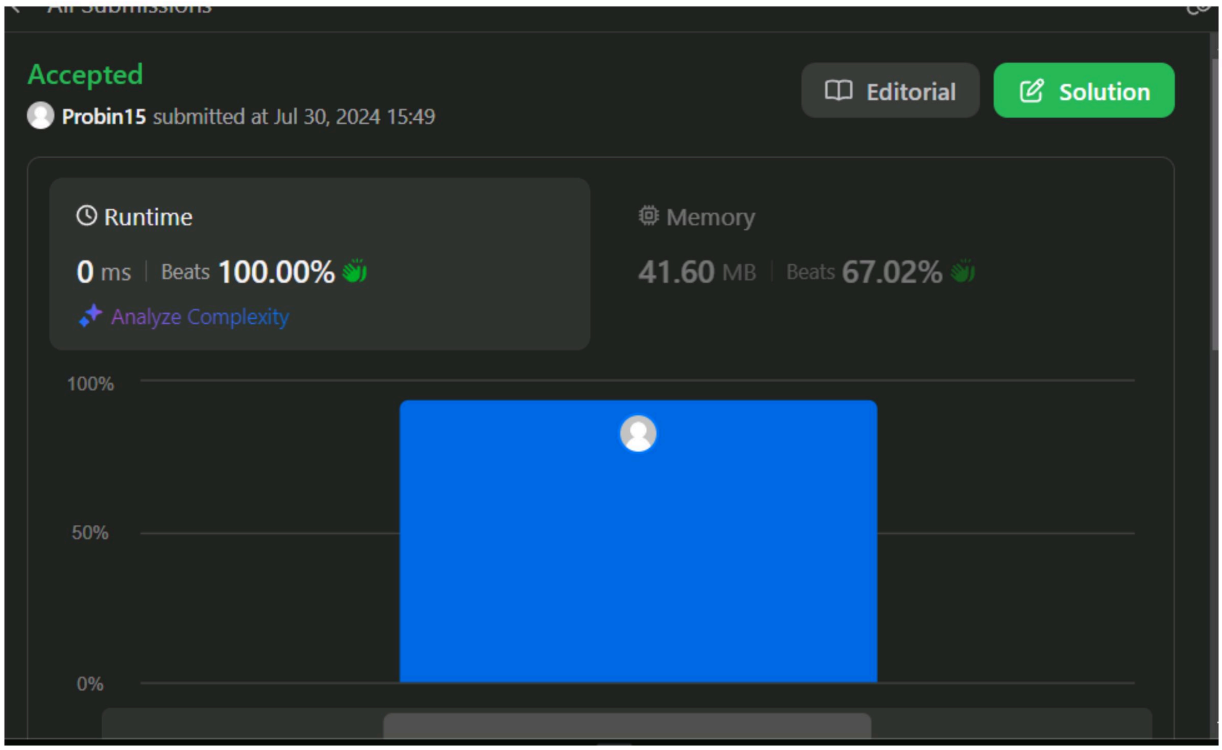
Increment the large integer by one and return *the resulting array of digits*.

Program Code:

```
class Solution {
    public int[] plusOne(int[] digits) {
     for (int i = digits.length - 1; i >= 0; i--) {
        if (digits[i] < 9){
            digits[i]++;
            return digits;
        }
```

```
        //last digit 0
        digits[i] = 0;
    }
        digits = new int[digits.length + 1];
        //first digit 1
        digits[0] = 1;
        return digits;
    }
}
```

OUTPUT :



PRACTICAL - 15

PROBLEM DEFINITION:

Given a non-negative integer x, return *the square root of* x *rounded down to the nearest integer*. The returned integer should be **non-negative** as well.

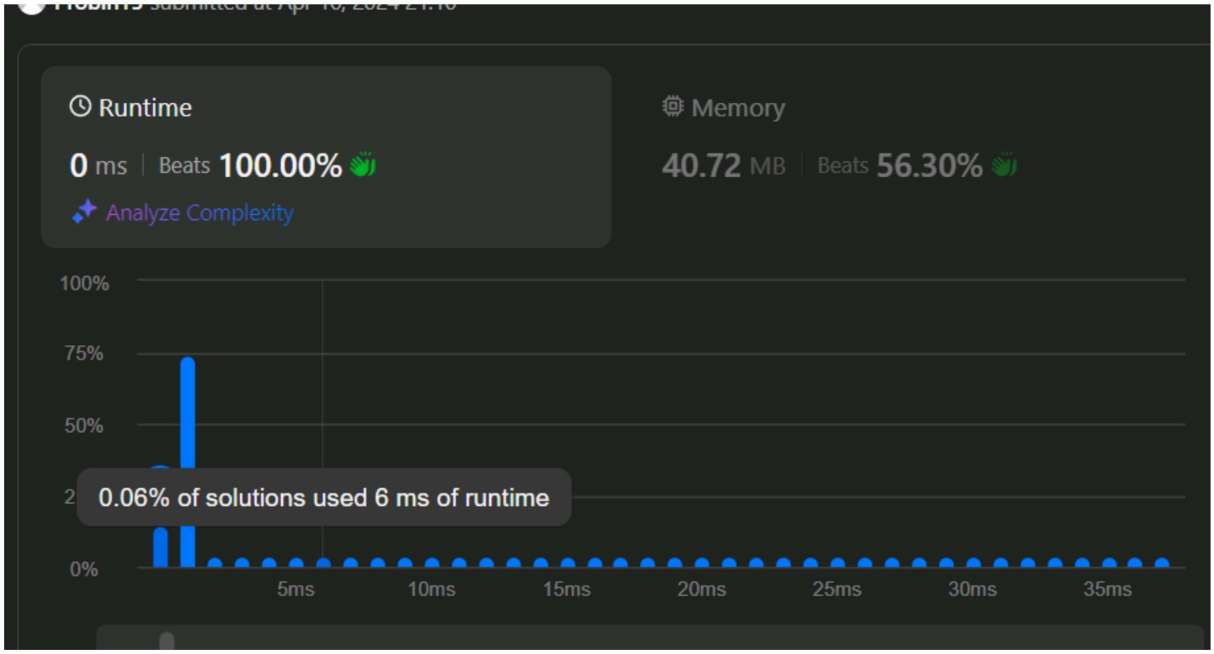
You **must not use** any built-in exponent function or operator.

- For example, do not use pow(x, 0.5) in c++ or x ** 0.5 in python.

Program Code:

```
class Solution {
    public int mySqrt(double x) {
        return (int) Math.sqrt(x);
    }
}
```

OUTPUT :



PRACTICAL - 16

PROBLEM DEFINITION:

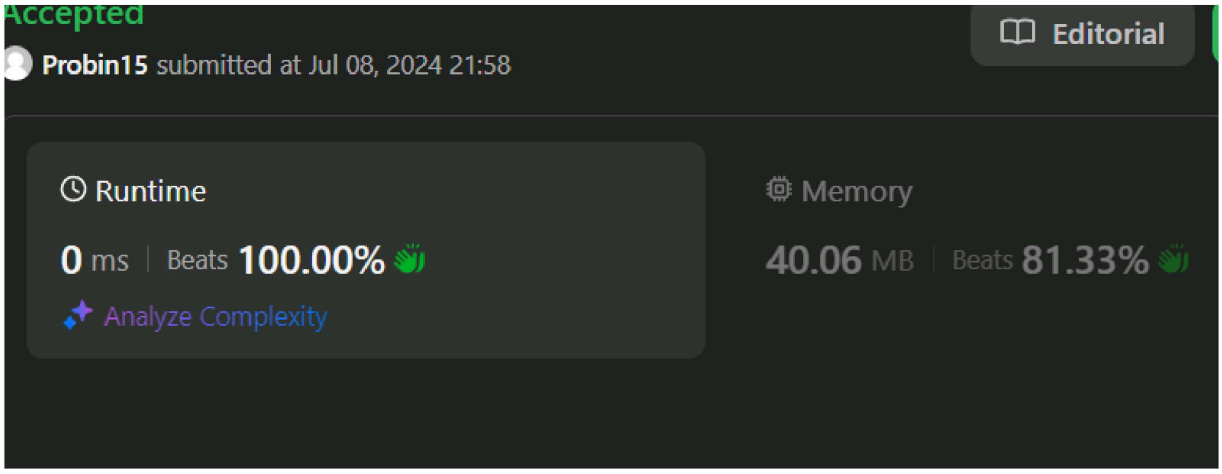
You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Program Code:

```
class Solution {
    public int climbStairs(int n) {
        //fibonacci recurrence relation
        int arr[] = new int[n + 1];
        arr[0] = 1;
        arr[1] = 1;

        for (int i = 2; i <= n; i++)
        arr[i] = arr[i - 1] + arr[i - 2];

        return arr[n];
    }
}
```

OUTPUT :



PRACTICAL - 17

PROBLEM DEFINITION:

Given an integer array nums of **unique** elements, return *all possible*

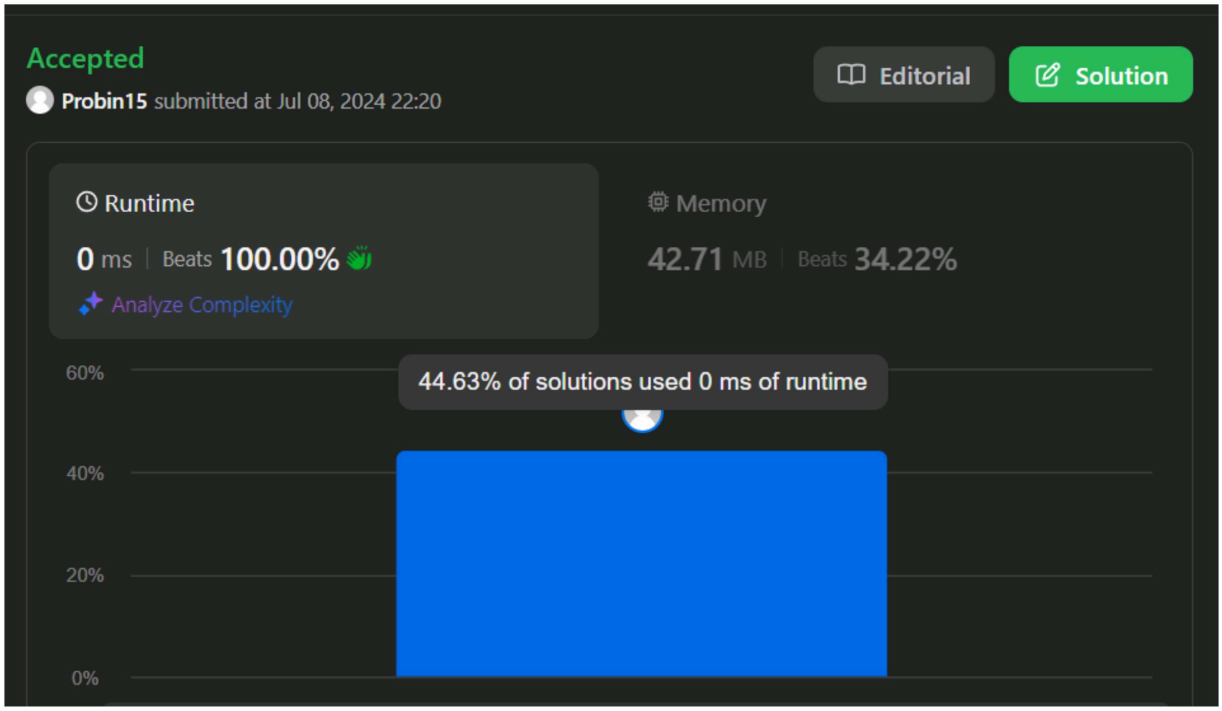
subsets
(the power set).

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Program Code:

```
class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        generateSubsets(nums, 0, new ArrayList<>(), result);
        return result;
    }

    public void generateSubsets(int[] nums, int index,List<Integer> current,List<List<Integer>> result)
{
        result.add(new ArrayList<>(current));

        for (int i = index; i < nums.length; i++) {
            current.add(nums[i]);
            generateSubsets(nums, i + 1, current, result);
            current.remove(current.size() - 1);
        }
    }

}
```

OUTPUT :



PRACTICAL - 18

PROBLEM DEFINITION:

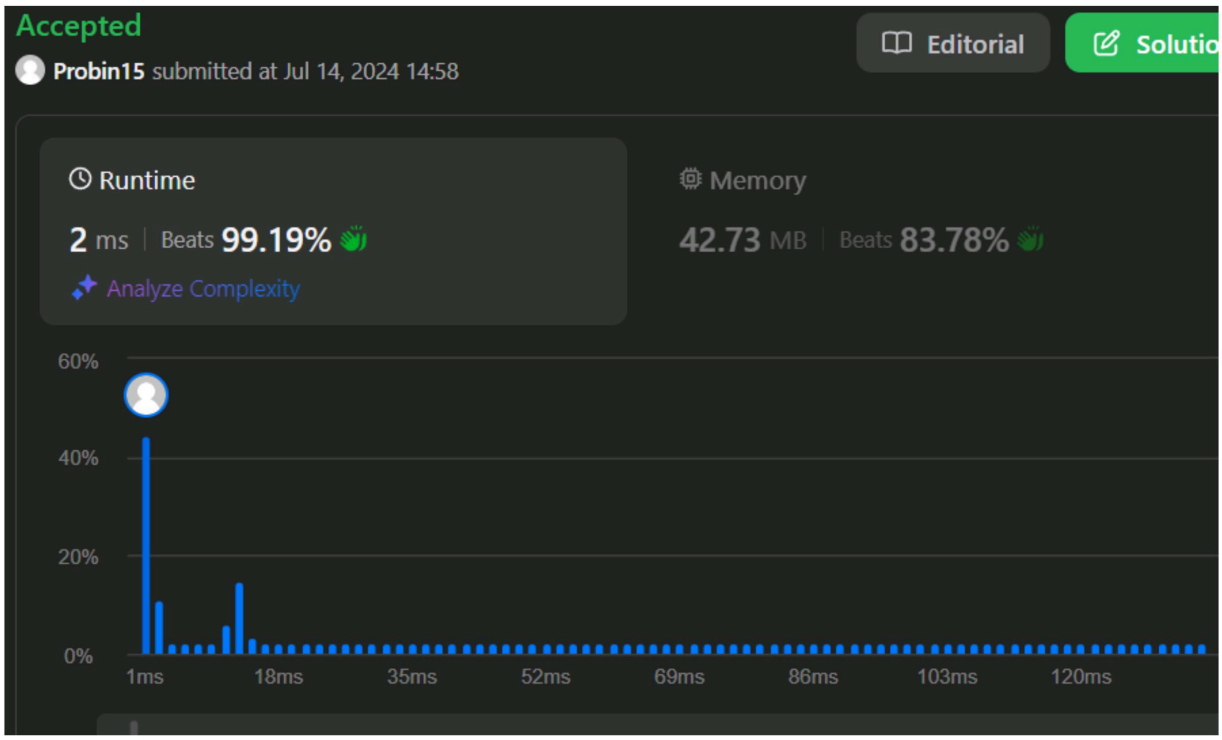
A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string s, return true *if it is a **palindrome**, or* false *otherwise*.

Program Code:

```
class Solution {
    public boolean isPalindrome(String s) {
        int i = 0;
        int j = s.length() - 1;
        while (i < j) {
            while (i < j && !Character.isLetterOrDigit(s.charAt(i))) {
                i++;
            }
            while (i < j && !Character.isLetterOrDigit(s.charAt(j))) {
                j--;
            }
            if (Character.toLowerCase(s.charAt(i)) != Character.toLowerCase(s.charAt(j))) {
                return false;
            }
            i++;
            j--;
        }
        return true;
    }
}
```

OUTPUT :



PRACTICAL - 19

PROBLEM DEFINITION:

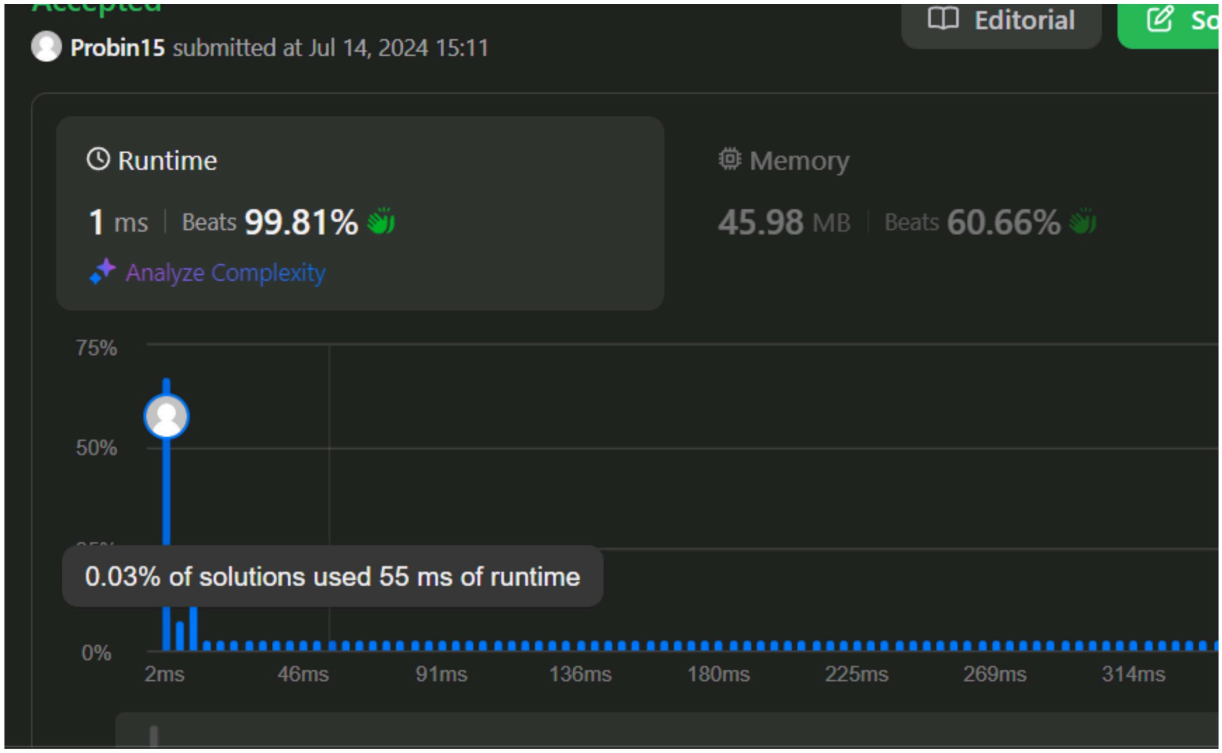
Given a **non-empty** array of integers nums, every element appears *twice* except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

Program Code:

```
class Solution {
    public int singleNumber(int[] nums) {
        int ans=0;
        for(int i=0 ; i<nums.length ; i++){
            ans=ans^nums[i];
        }
        return ans;
    }
}
```

OUTPUT :



PRACTICAL - 20

PROBLEM DEFINITION:

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

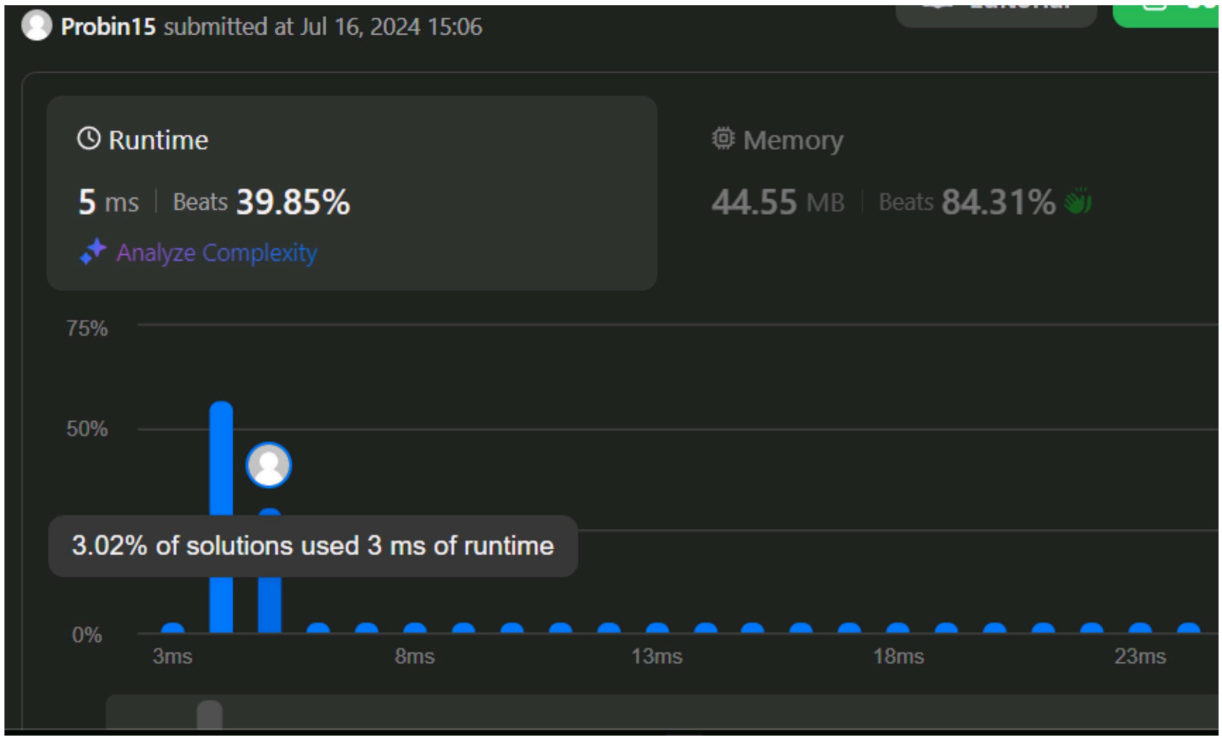
- MinStack() initializes the stack object.
- void push(int val) pushes the element val onto the stack.
- void pop() removes the element on the top of the stack.
- int top() gets the top element of the stack.
- int getMin() retrieves the minimum element in the stack.

You must implement a solution with O(1) time complexity for each function.

Program Code:

```
class MinStack {
    public Stack<Integer> dataStack;
    public Stack<Integer> minStack;

    public MinStack() {
        dataStack = new Stack<>();
        minStack = new Stack<>();
    }
    public void push(int val) {
        dataStack.push(val);
        if (minStack.isEmpty() || val <= minStack.peek()) {
            minStack.push(val);
        }
    }
    public void pop() {
        if (!dataStack.isEmpty()) {
            int removed = dataStack.pop();
            if (removed == minStack.peek()) {
                minStack.pop();
            }
        }
    }
    //
    public int top() {
        return dataStack.peek();
    }
    //peek function
    public int getMin() {
        return minStack.peek();
    }
    //minimum element from the stack
}
```

OUTPUT :


Probin15 submitted at Jul 16, 2024 15:06
Runtime
5 ms | Beats 39.85%
Analyze Complexity
Memory
44.55 MB | Beats 84.31%
75%
50%
3.02% of solutions used 3 ms of runtime
0%
3ms
8ms
13ms
18ms
23ms

PRACTICAL - 21

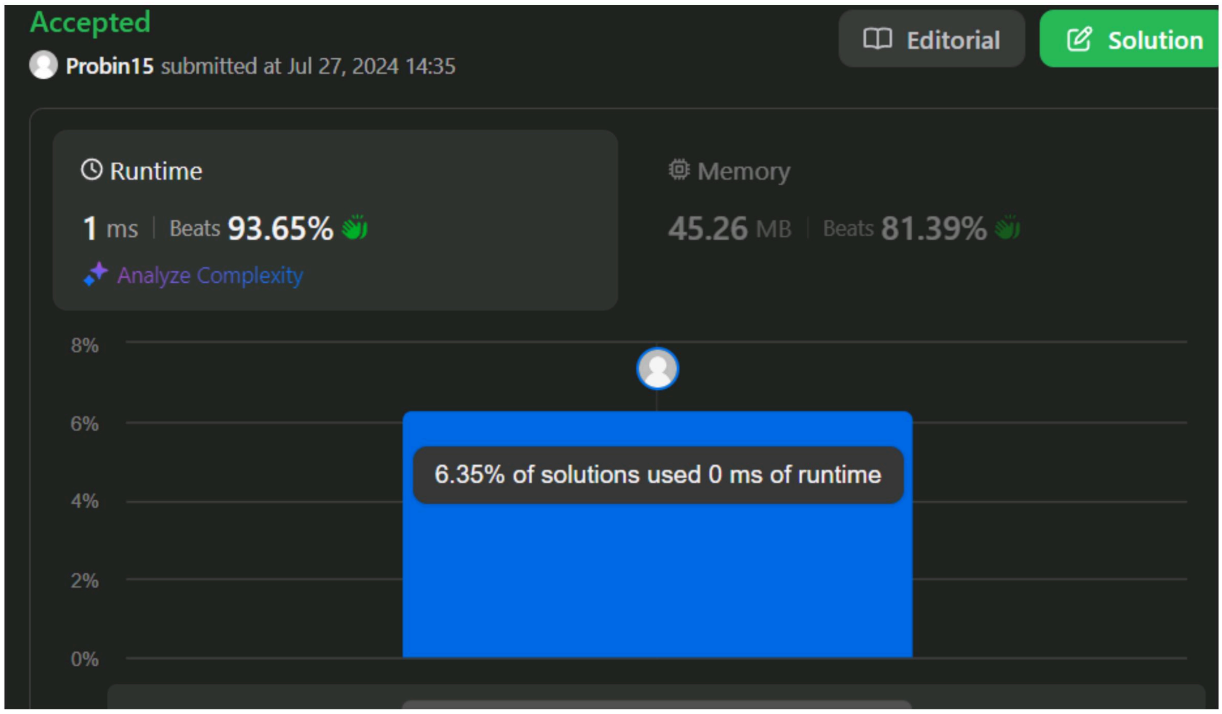
PROBLEM DEFINITION:

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return *the new head*.

Program Code:

```
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        //head case
        while(head!=null && head.val==val){
            head=head.next;
        }
        //normal case
        ListNode temp=head;
        while(temp!=null && temp.next!=null){
            if(temp.next.val==val){
                temp.next=temp.next.next;
            }
            else{
                temp=temp.next;
            }
        }
        return head;
    }
}
    }
};
```

OUTPUT :


Accepted
Probin15 submitted at Jul 27, 2024 14:35
Editorial
Solution
Runtime
1 ms | Beats 93.65%
Analyze Complexity
Memory
45.26 MB | Beats 81.39%
8%
6%
4%
2%
0%
6.35% of solutions used 0 ms of runtime

PRACTICAL - 22

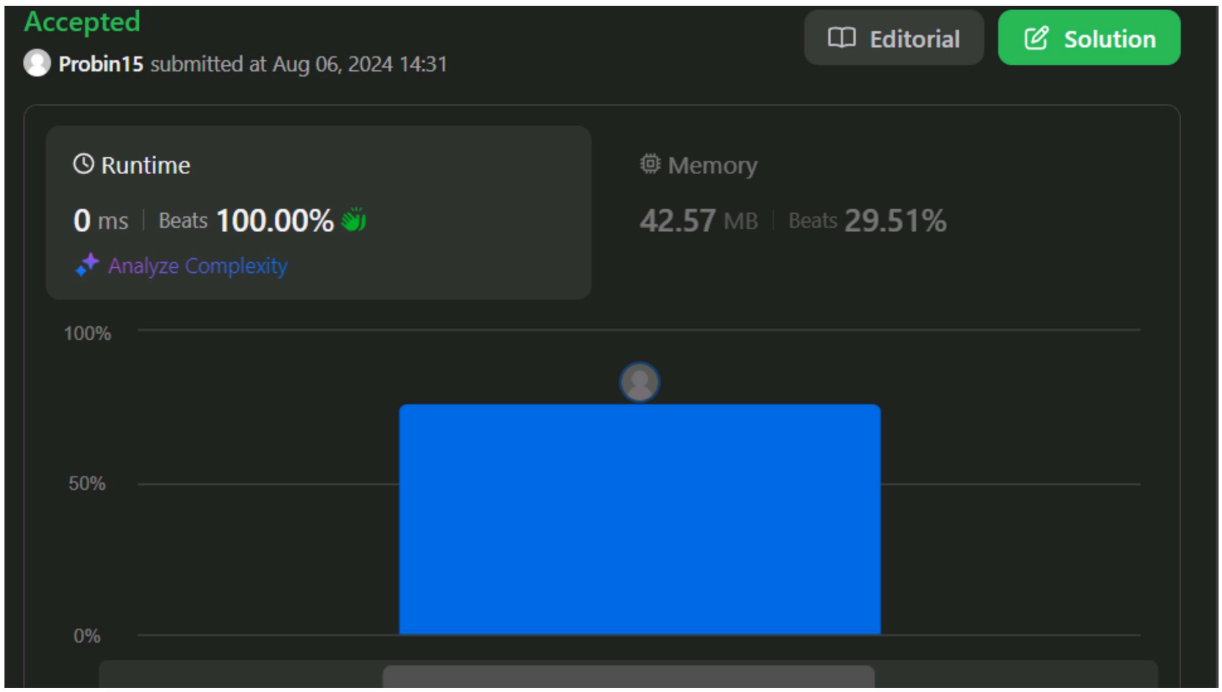
PROBLEM DEFINITION:

Given the head of a singly linked list, reverse the list, and return *the reversed list*.

Program Code:

```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        while(head!=null){
            ListNode temp = head.next;
            head.next = prev;
            prev = head;
            head = temp;
        }
        return prev;
        //reversed traversal of linkedlist
    }
}
```

OUTPUT :



PRACTICAL - 23

PROBLEM DEFINITION:

Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

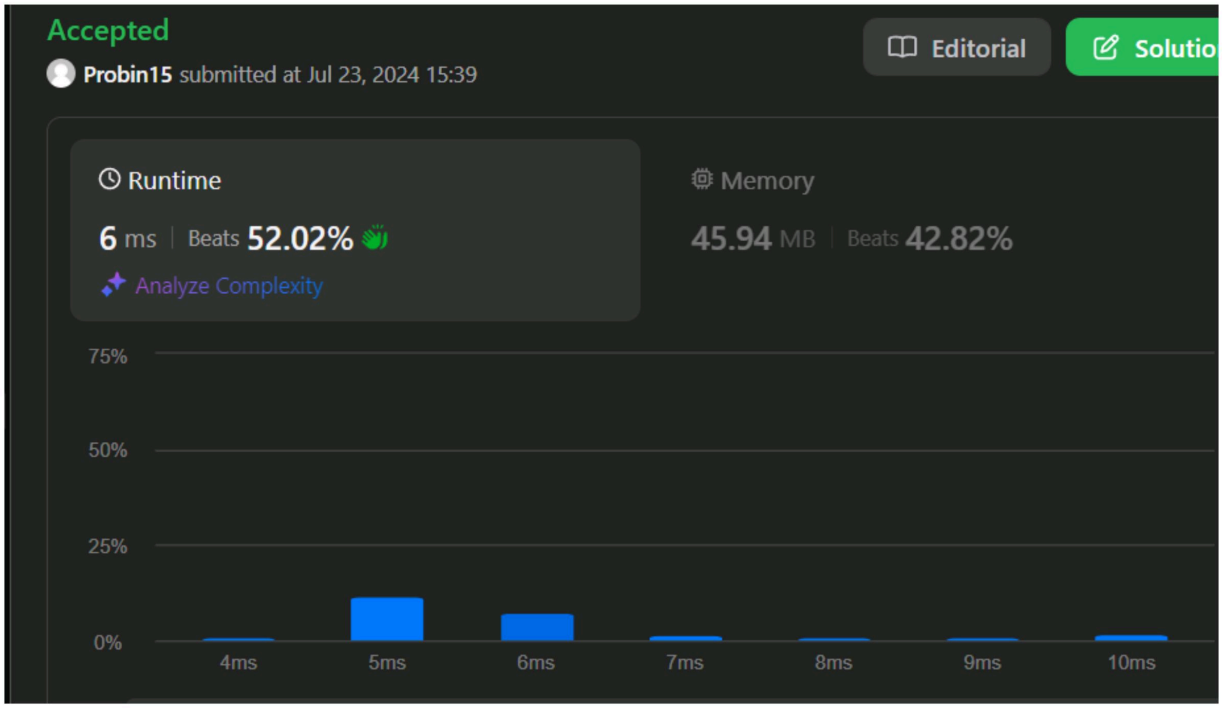
- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

Program Code:

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int row = 0;
        int col = matrix[0].length - 1;

        while (row < matrix.length && col >= 0) {
            if (matrix[row][col] == target) {
                return true;
            } else if (target < matrix[row][col]) {
                col--;
            } else {
                row++;
            }
```

```
        }

        return false;
    }
}
```

OUTPUT :



PRACTICAL - 24

PROBLEM DEFINITION:

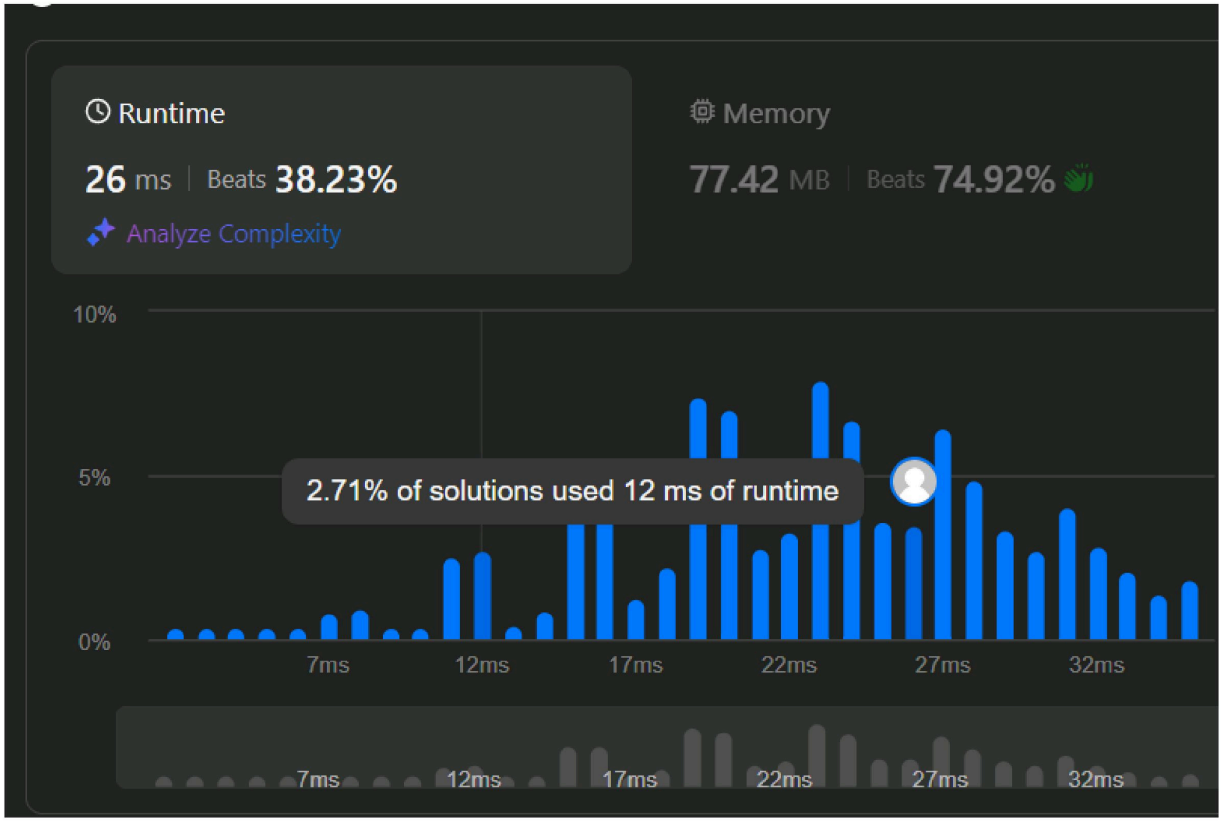
You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Program Code:

```cpp
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode* dummyHead = new ListNode(0);
        ListNode* tail = dummyHead;
        int carry = 0;

        while (l1 != nullptr || l2 != nullptr || carry != 0) {
            int digit1 = (l1 != nullptr) ? l1->val : 0;
            int digit2 = (l2 != nullptr) ? l2->val : 0;

            int sum = digit1 + digit2 + carry;
            int digit = sum % 10;
            carry = sum / 10;

            ListNode* newNode = new ListNode(digit);
            tail->next = newNode;
            tail = tail->next;

            l1 = (l1 != nullptr) ? l1->next : nullptr;
            l2 = (l2 != nullptr) ? l2->next : nullptr;
        }

        ListNode* result = dummyHead->next;
        delete dummyHead;
        return result;
    }
};
```

OUTPUT :



PRACTICAL - 25

PROBLEM DEFINITION:

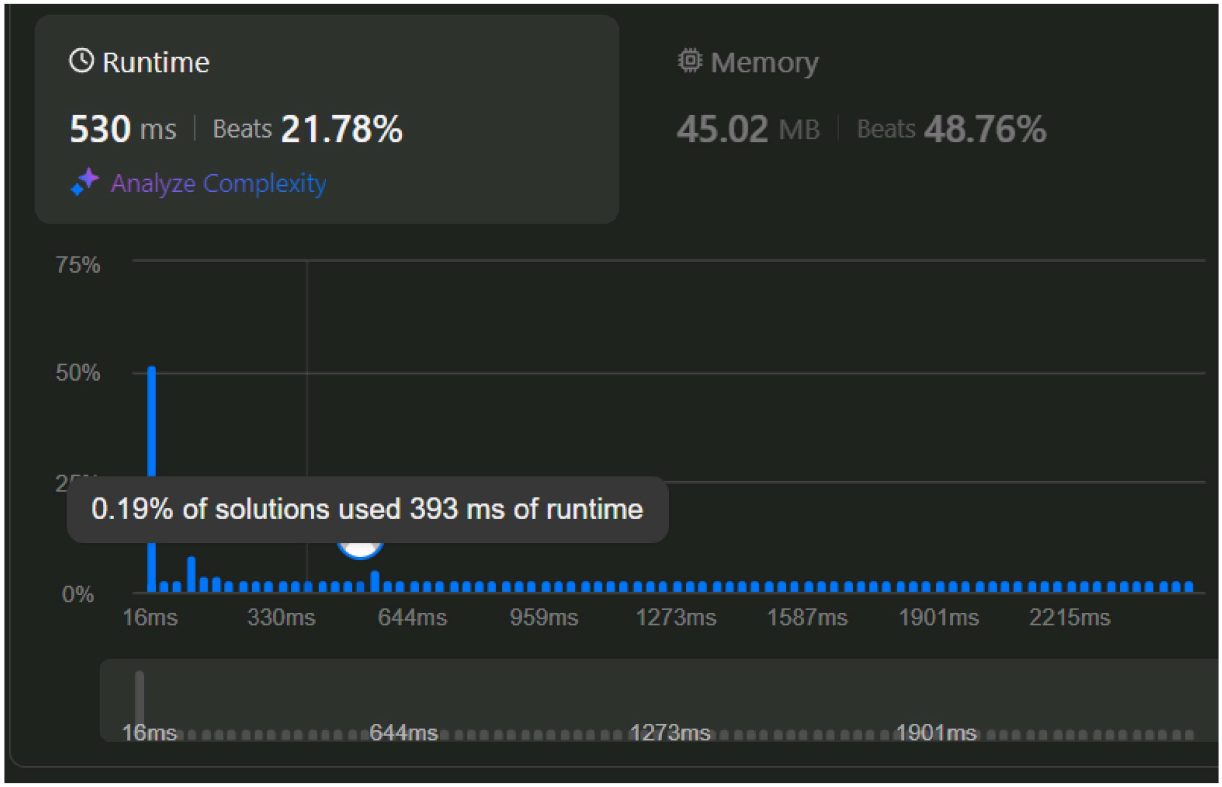
Given a string s, return *the longest palindromic substring* in s.

Program Code:

```
public class Solution {
    public String longestPalindrome(String s) {
        if (s.length() <= 1) {
            return s;
        }

        int maxLen = 1;
        String maxStr = s.substring(0, 1);

        for (int i = 0; i < s.length(); i++) {
```

```
            for (int j = i + maxLen; j <= s.length(); j++) {
                if (j - i > maxLen && isPalindrome(s.substring(i, j))) {
                    maxLen = j - i;
                    maxStr = s.substring(i, j);
                }
            }
        }

        return maxStr;
    }

    private boolean isPalindrome(String str) {
        int left = 0;
        int right = str.length() - 1;

        while (left < right) {
            if (str.charAt(left) != str.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }

        return true;
    }
}
```

OUTPUT :



PRACTICAL - 26

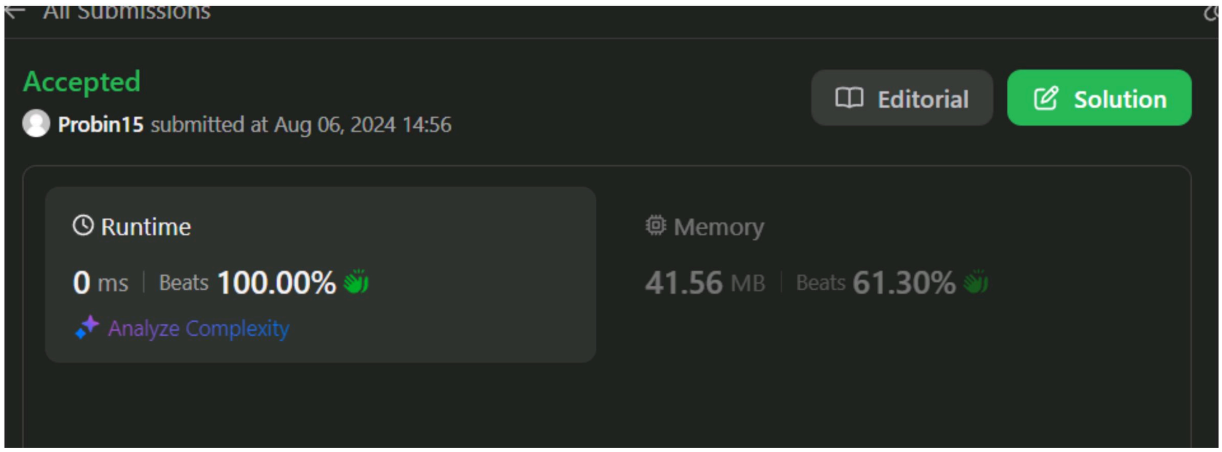
PROBLEM DEFINITION:

Given a sorted array that is rotated at an unknown pivot, search for a target value. If the target is found, return its index; otherwise, return -1. The array has no duplicate elements.

Program Code:

```
class Solution{

    public ListNode removeNthFromEnd(ListNode head, int n) {
        if (head.next == null)
            return null;

        int size = 0;
        ListNode temp = head;
        while (temp != null) {
            size++;
            temp = temp.next;
        }

        if (n == size) {
            head = head.next;
            return head;
        }

        ListNode prev = head;
        for (int i = 1; i < size-n; i++) {
            prev = prev.next;
        }

        ListNode delnode = prev.next;

        if (delnode.next == null){
            prev.next = null;
        }
        else{
            prev.next = delnode.next;
        }

        return head;
    }
}
```

OUTPUT :



PRACTICAL - 27

PROBLEM DEFINITION:

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the i^{th} line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store*.

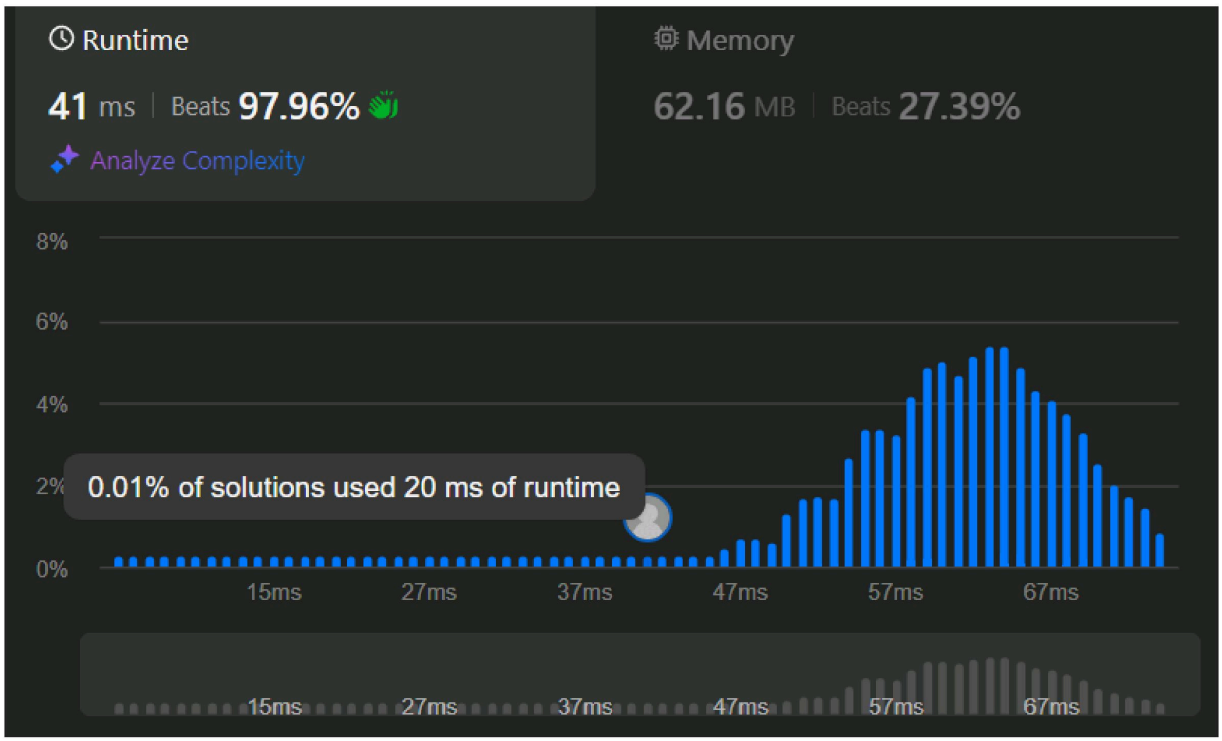
Notice that you may not slant the container.

Program Code:

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
class Solution {
public:
    int maxArea(vector<int>& height) {
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);
        cout.tie(NULL);
        int ans = 0;
        int left = 0;
        int right = height.size() - 1;
```

```
        while (left < right) {
            int width = right - left;
            int currentHeight = min(height[left], height[right]);
            int currentArea = width * currentHeight;
            ans = max(ans, currentArea);

            if (height[left] < height[right]) {
                left++;
            } else {
                right--;
            }
        }

        return ans;
    }
};
```

OUTPUT :



PRACTICAL - 28

PROBLEM DEFINITION:

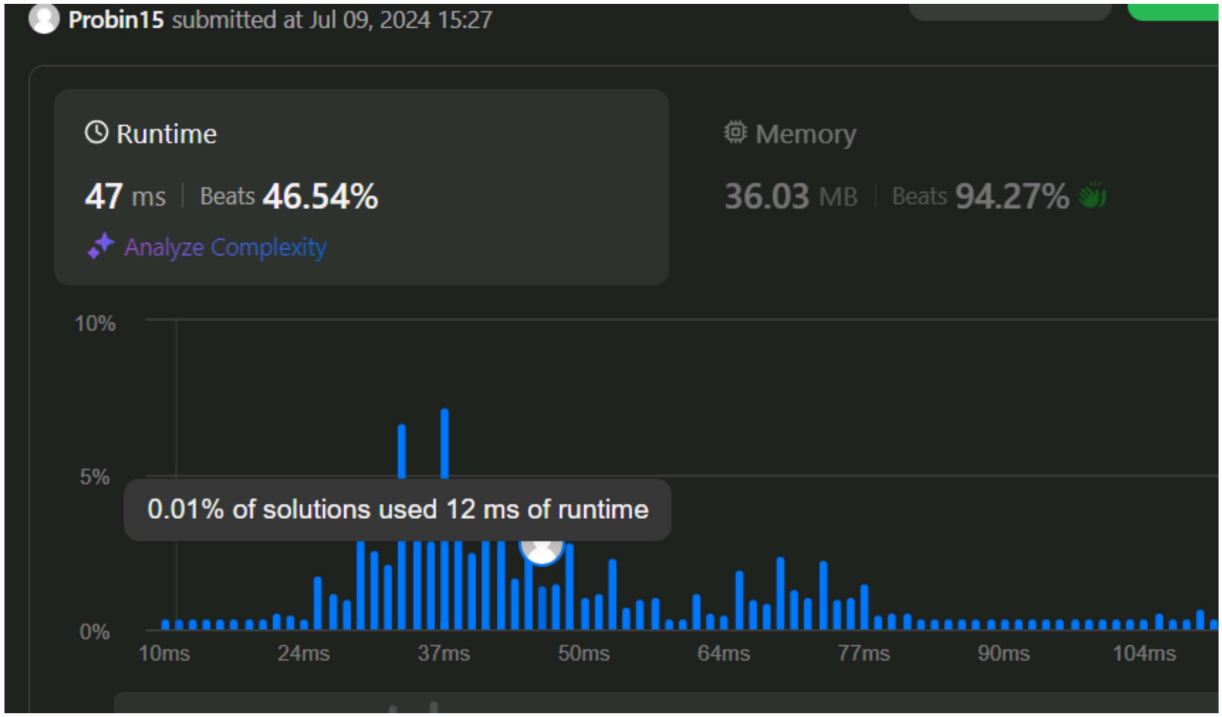
Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears **at most twice**, return *an array of all the integers that appears **twice***.

You must write an algorithm that runs in O(n) time and uses only *constant* auxiliary space, excluding the space needed to store the output

Program Code:

```
class Solution {
public:
    vector<int> findDuplicates(vector<int>& nums) {
        vector<int>result;
        for(int i=0 ; i<nums.size() ; i++){
            int index=abs(nums[i])-1;
            if(nums[index]<0){
                result.push_back(abs(nums[i]));
            }
            else{
                nums[index]=-nums[index];
            }
        }
        return result;
    }

};
```

OUTPUT :



PRACTICAL - 29

PROBLEM DEFINITION: A perfect number is a **positive integer** that is equal to the sum of its **positive divisors**, excluding the number itself. A **divisor** of an integer x is an integer that can divide x evenly.

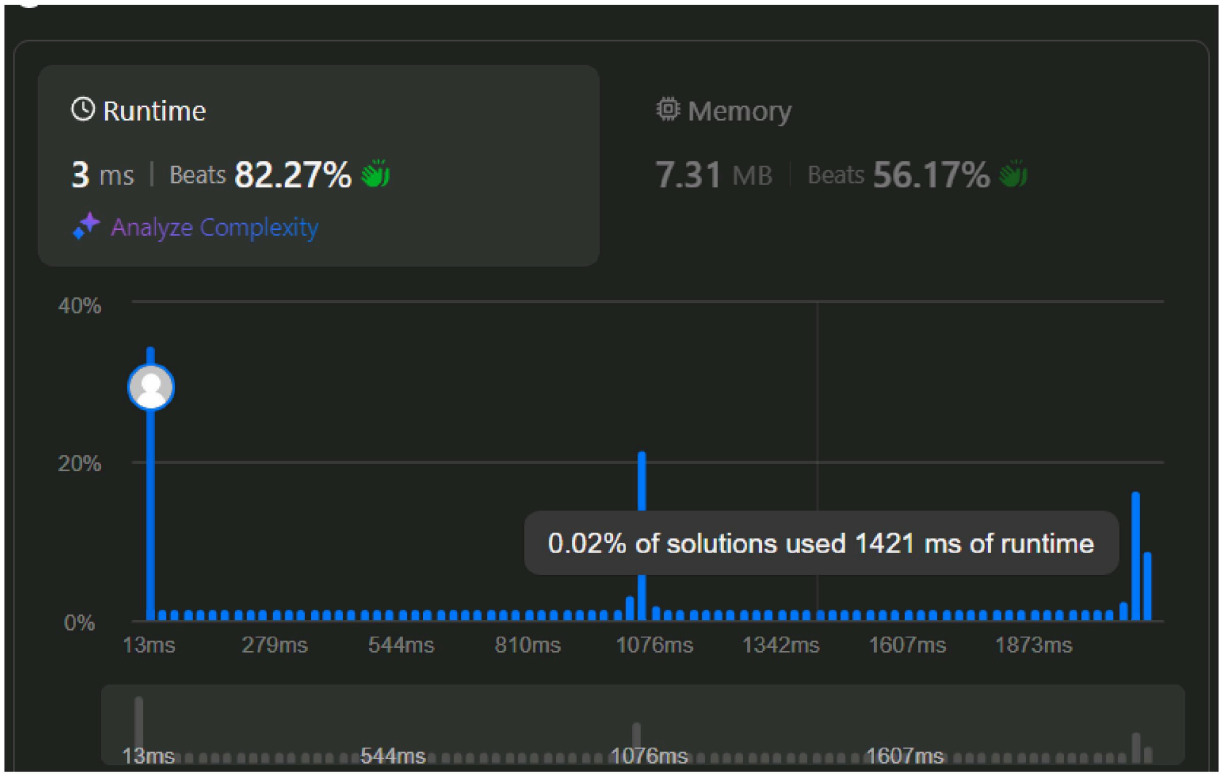
Given an integer n, return true *if* n *is a perfect number, otherwise return* false.

Program Code:

```
class Solution {
public:
    bool checkPerfectNumber(int num) {
        if (num == 1) return false;
        int sum = 1;  // 1 is a divisor of every number
        for (int i = 2; i * i <= num; i++) {
            if (num % i == 0) {
                sum += i;
                if (i != num / i) sum += num / i;  // Add both divisors
            }
```

```
        }
        return sum == num;
    }
};
```

OUTPUT :



PRACTICAL - 30

PROBLEM DEFINITION:

The **Fibonacci numbers**, commonly denoted F(n) form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

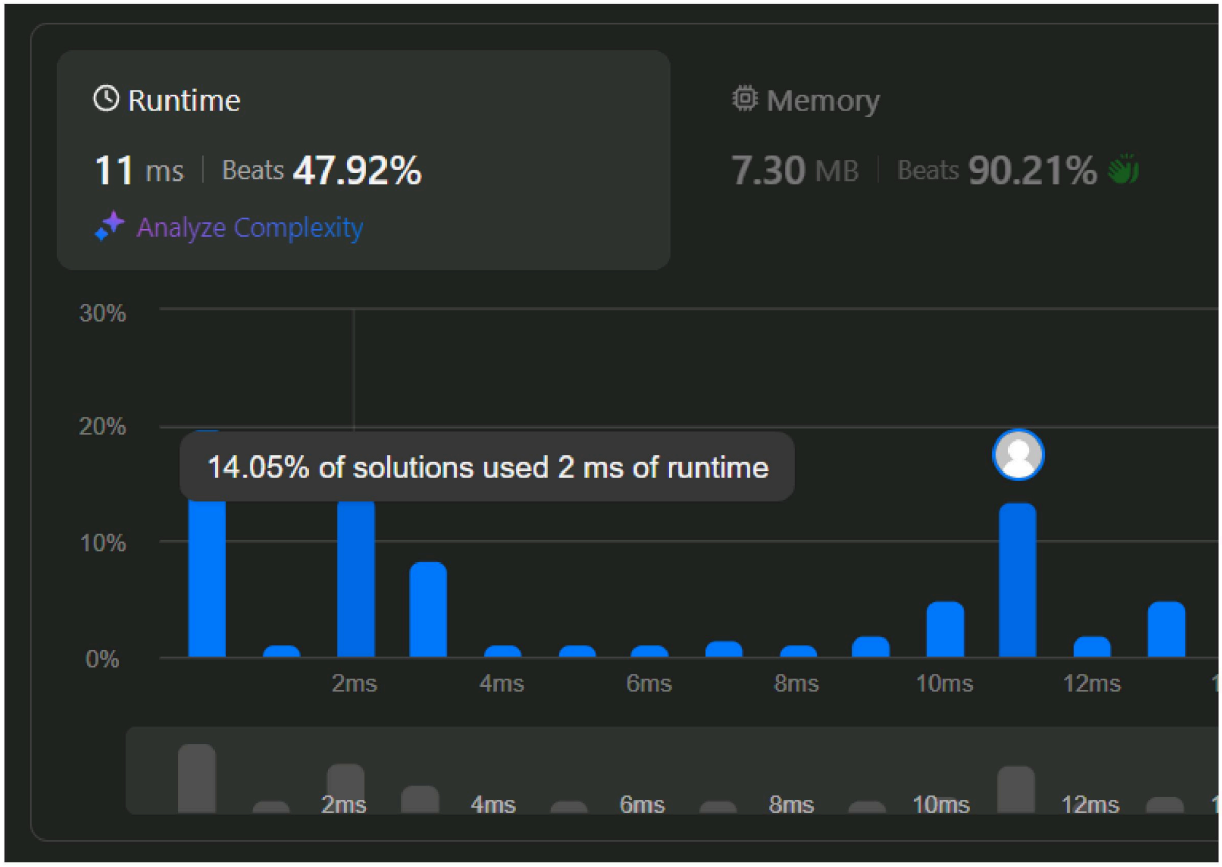
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.

Given n, calculate F(n).

Program Code:

```
class Solution {
public:
    int fib(int n) {
        if (n == 0)
            return 0;

        if (n == 1)
            return 1;

        int ans = fib(n - 1) + fib(n - 2);

        return ans;
    }
};
```

OUTPUT :


Runtime
11 ms | Beats 47.92%
Analyze Complexity
Memory
7.30 MB | Beats 90.21%
30%
20%
10%
0%
14.05% of solutions used 2 ms of runtime
2ms
4ms
6ms
8ms
10ms
12ms